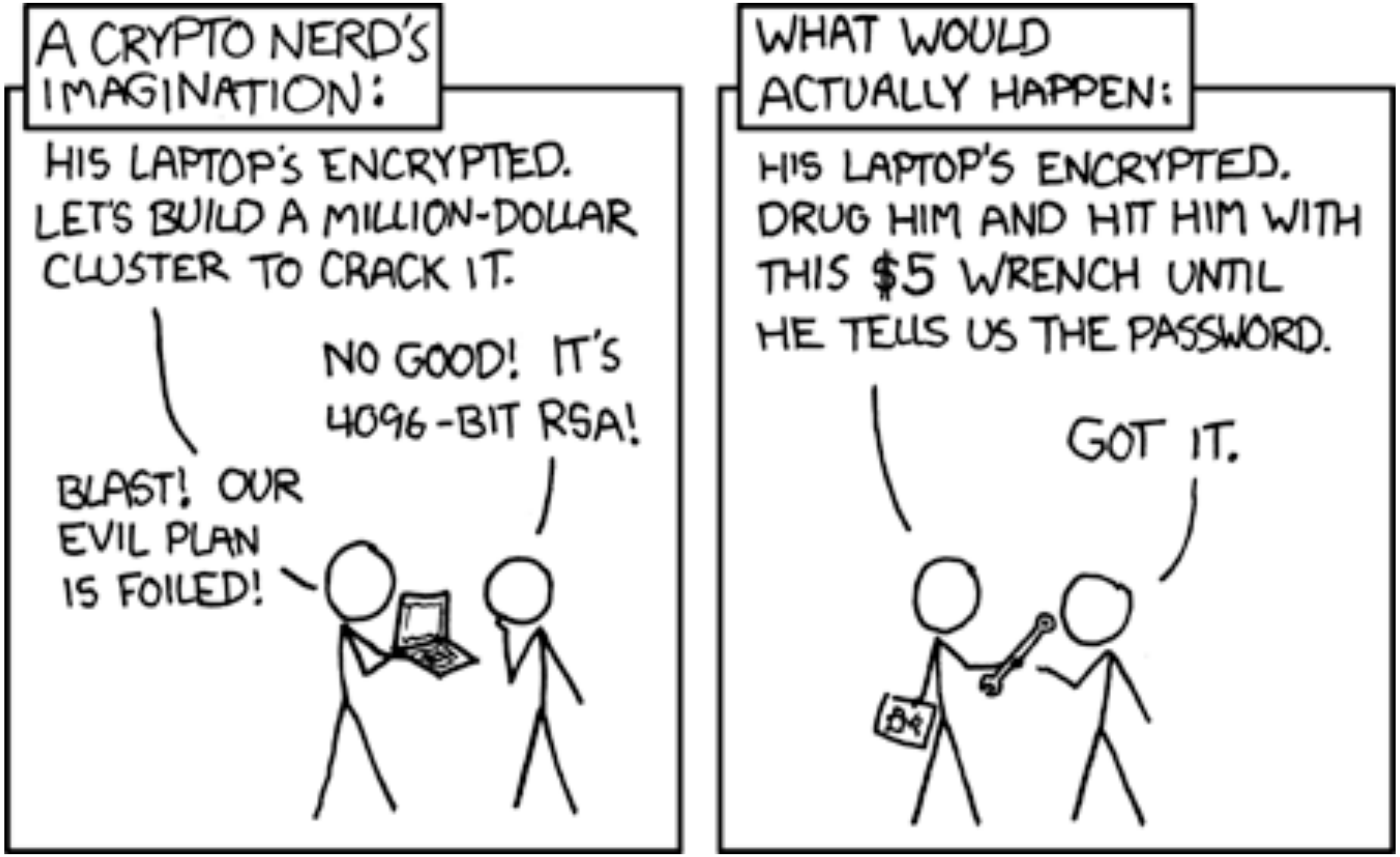
CS 88: Security and Privacy

11: Introduction to Cryptography

10-06-2022

slides adapted from Dave Levine, Vitaly Shmatikov, Christo Wilson, and Franzi Roesner

SWARTHMORE COLLEGE



XKCD: http://xkcd.com/538/

Cryptography

- Cryptography: An ancient art
  - 500BC – 20th century: Design -> break -> repair -> break -> repair ->.....
- Modern Cryptography: Cryptography as a ***science***
  - relies on rigorous threat models
  - firm theoretical foundations and proofs!

Modern Cryptography

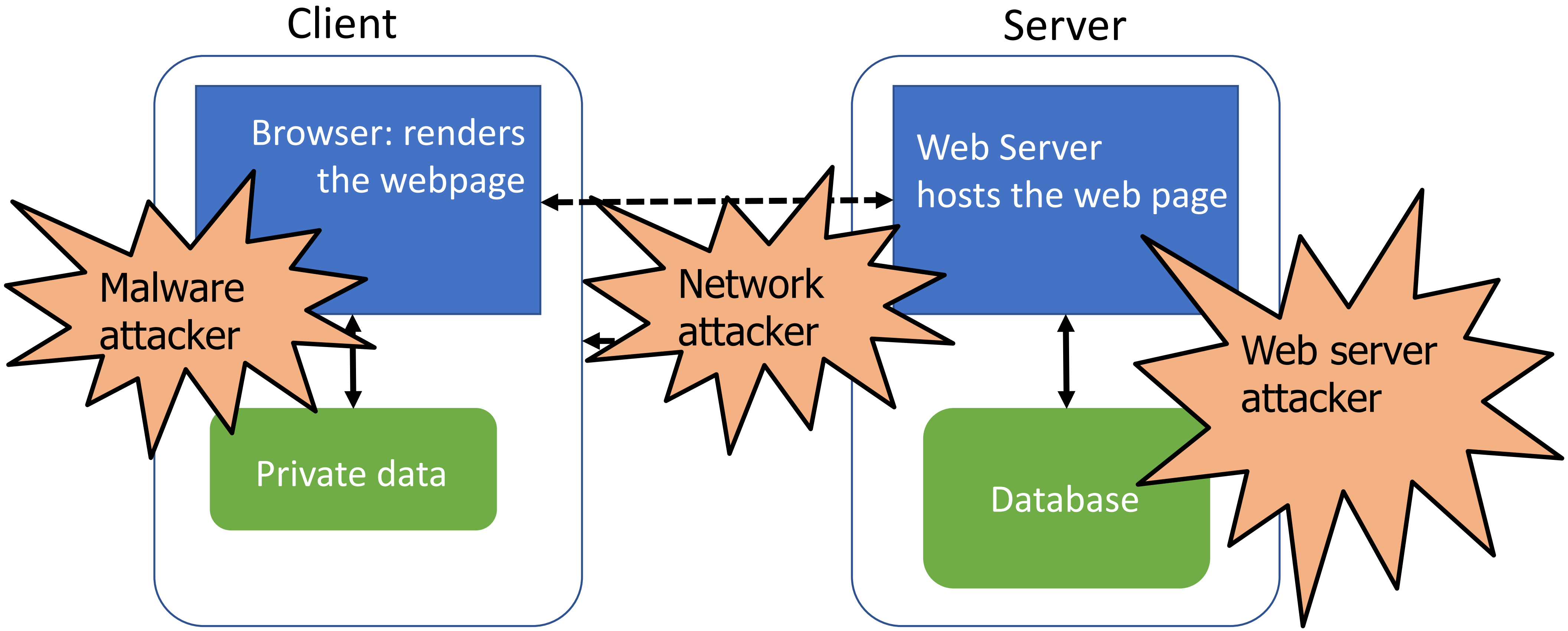
Design, analysis and implementation of mathematical techniques for securing information, systems and computation against adversarial attacks.

Modern Cryptography: How many of the following actions involve cryptography ?

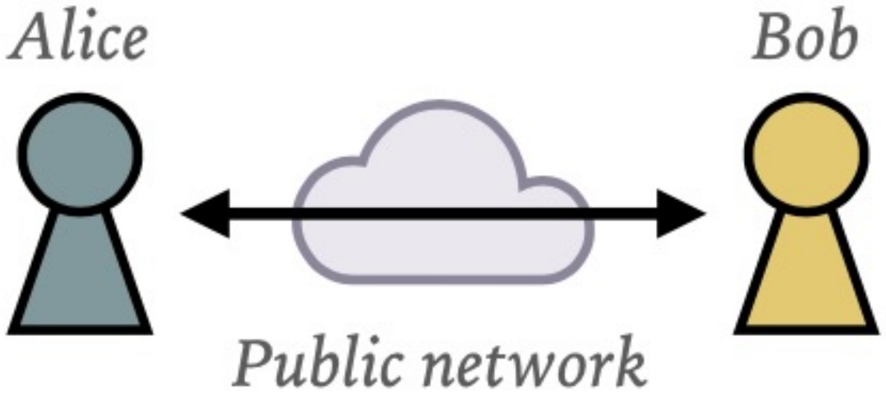
1. Git cloning your lab repo
2. Connecting to Swarthmore's WiFi
3. Updating software on your device
4. Making online purchases

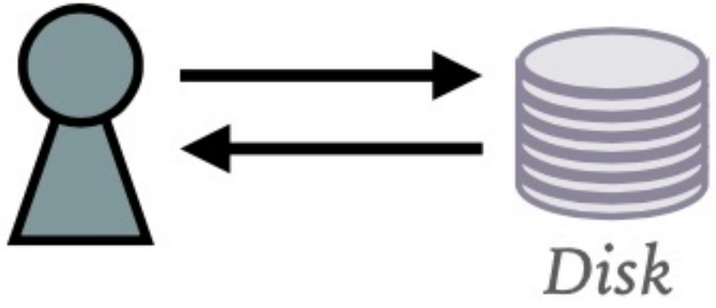
A. 1   B. 2   C. 3   D. 4 E. 0

Where Does the Attacker Live?

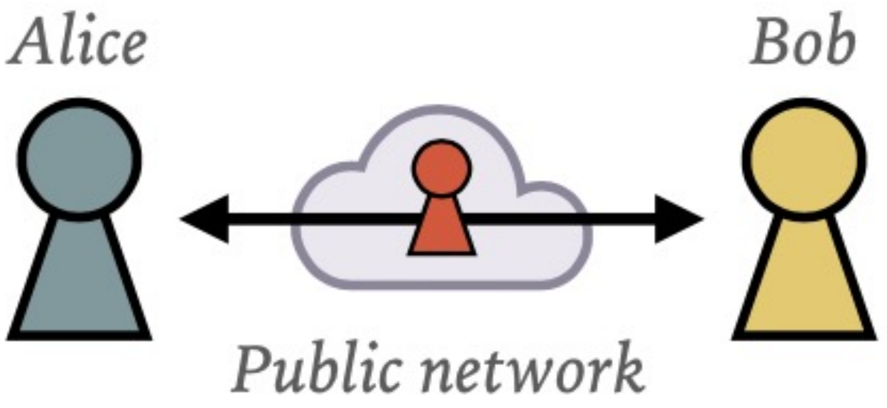


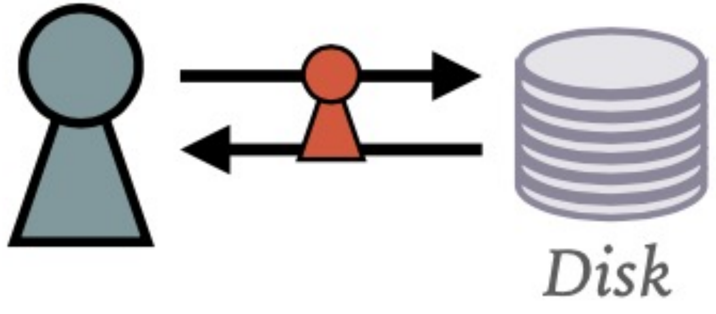
Scenarios and Goals





Scenarios and Goals

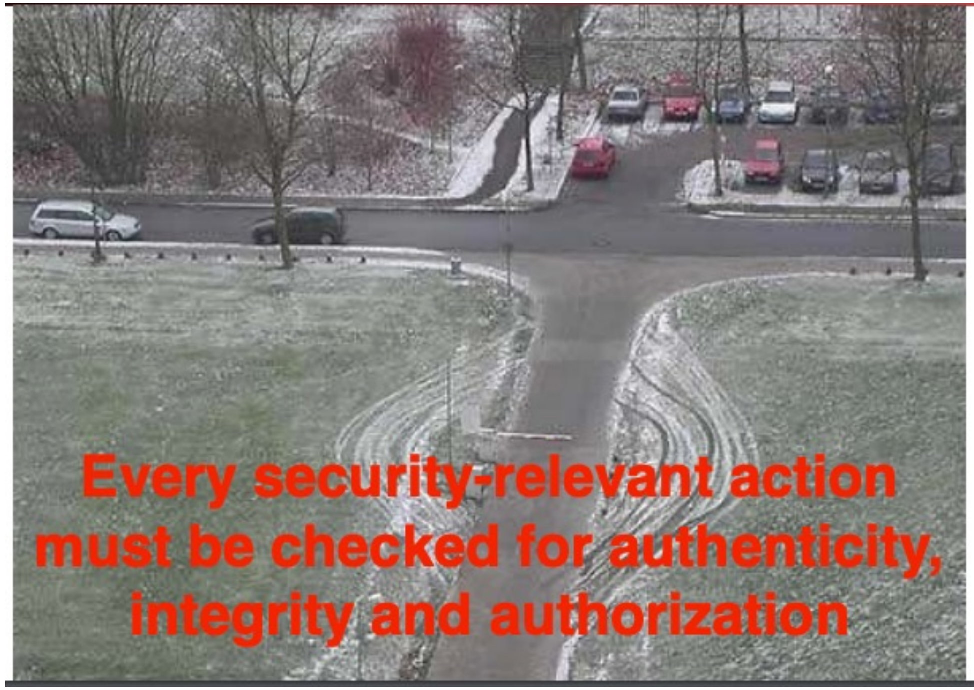




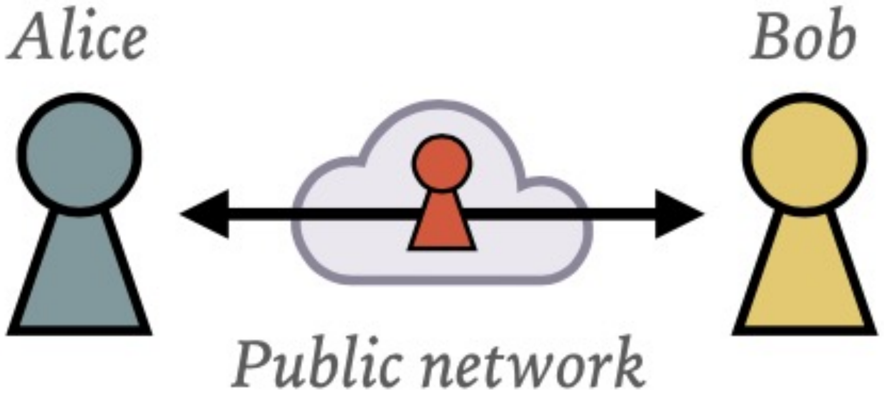
Confidentiality	Keep others from reading Alice's messages/data
Integrity	Keep others from undetectably tampering with Alice's messages/data
Authenticity	Keep others from undetectably impersonating Alice (keep her to her word too!)

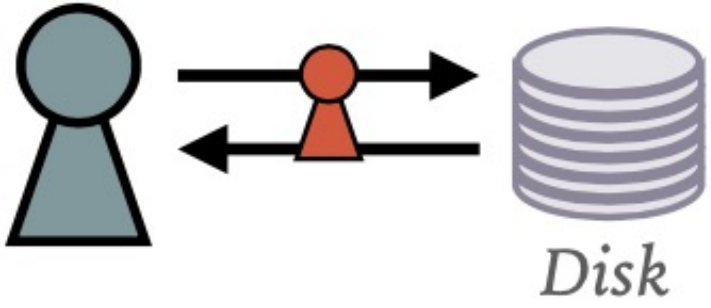
Recall the Bigger Picture

- Cryptography: small piece of a larger system
- Protect the entire system (recall: the weakest link)
  - physical security
  - OS security
  - Network security
  - Users
  - Cryptography
- Cryptography is a crucial part of this toolbox



Cryptography: Terms





Encryption (E): The process of transforming a message so that its meaning is not obvious
Decryption (D): The process of transforming an encrypted message back into its original form.
Plaintext (P): Original, unencrypted form of a message
Ciphertext (C): The encrypted form of a message

Formal Notation: We seek a cryptosystem for which P = D (E (P))

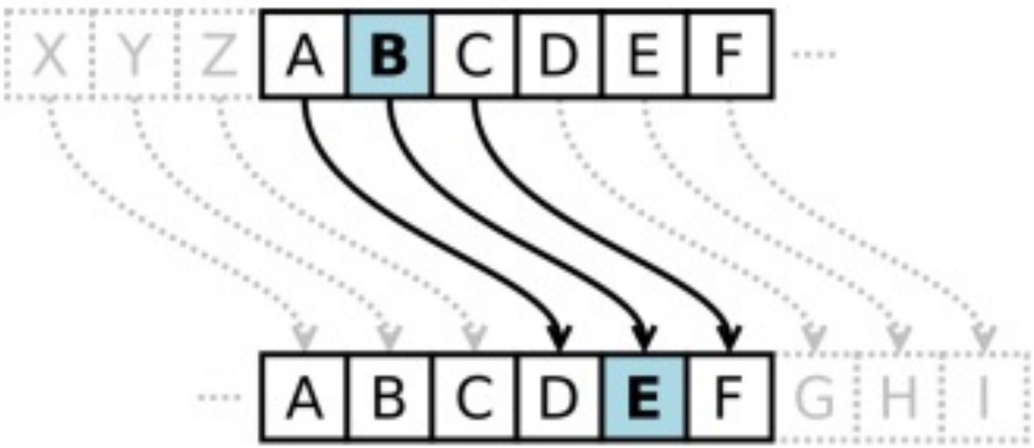
Historical Ciphers

- Substitution Cipher
  - Monoalphabetic - Ceasar's Cipher – fixed subst. over the entire message
  - Polyalphabetic – a number of substitutions at different positions in the message
- Transposition Ciphers
- Codebooks
- Machines

Recommended Reading: The Codebreakers by David Kahn, The Code Book by Simon Singh

Ceasar Cipher: Substitution Cipher

Plaintext letters replaced with letters fixed shift way in the alphabet.

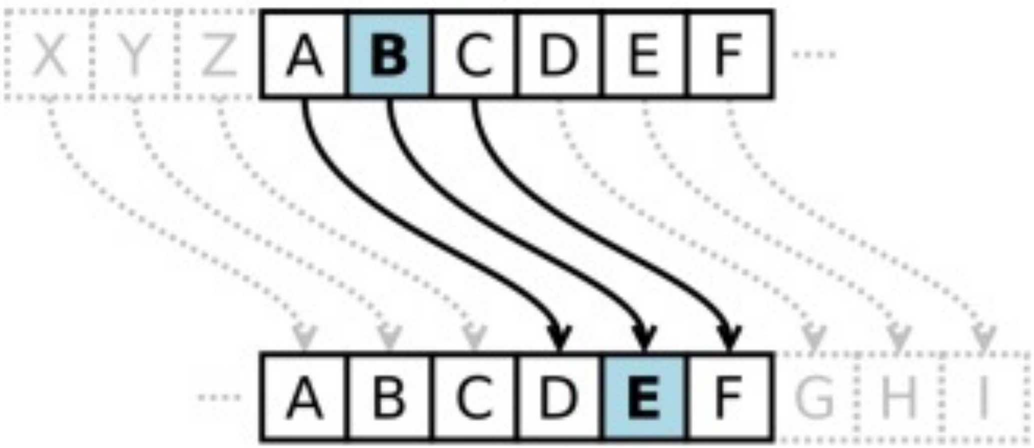


Example:

- Plaintext: HEY BRUTUS BRING A KNIFE TO THE PARTY.
- Ciphertext: KHB EUXWXV EULQJ D NQLIH WR WKH SDUWB
- Key Shift 3:
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - DEFGHIJKLMNOPQRSTUVWXYZABC

Ceasar Cipher: Substitution Cipher

Plaintext letters replaced with letters fixed shift way in the alphabet.



Example:

- Plaintext: HEY BRUTUS BRING A KNIFE TO THE PARTY.
- Ciphertext: KHB EUXWXV EULQJ D NQLIH WR WKH SDUWB
- Key Shift 3:
  - ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - DEFGHIJKLMNOPQRSTUVWXYZABC

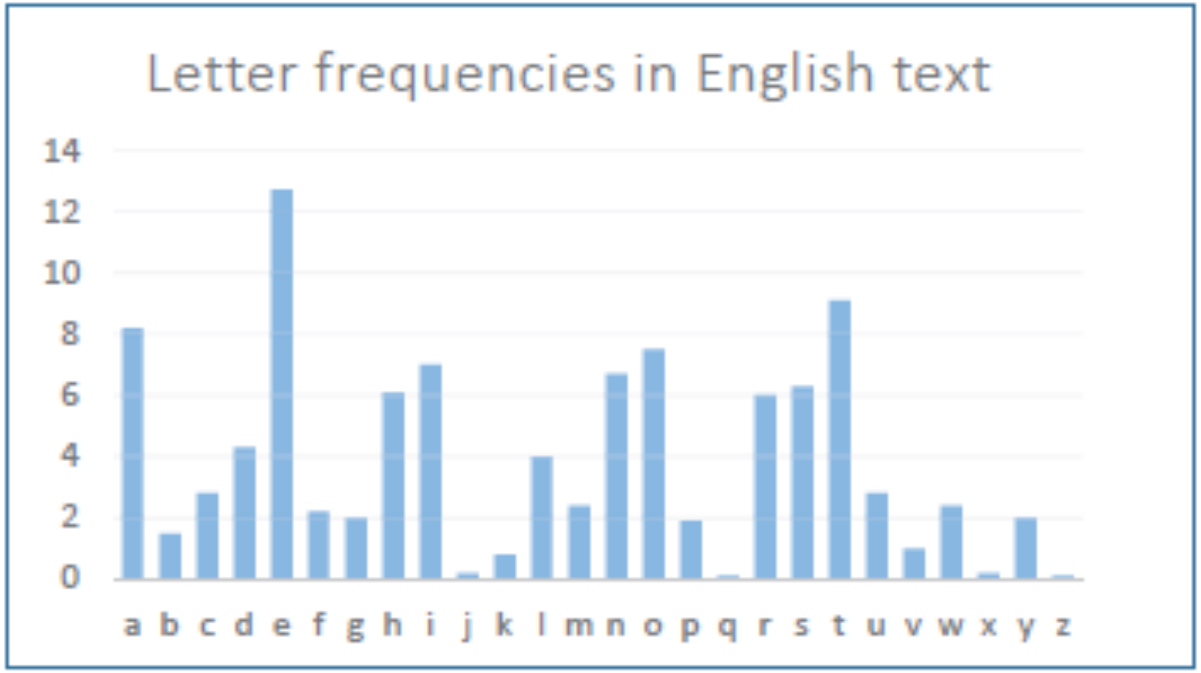
- Encryption and Decryption are symmetric.
- Key space?
  - 26
- Attack shift ciphers?
  - brute force

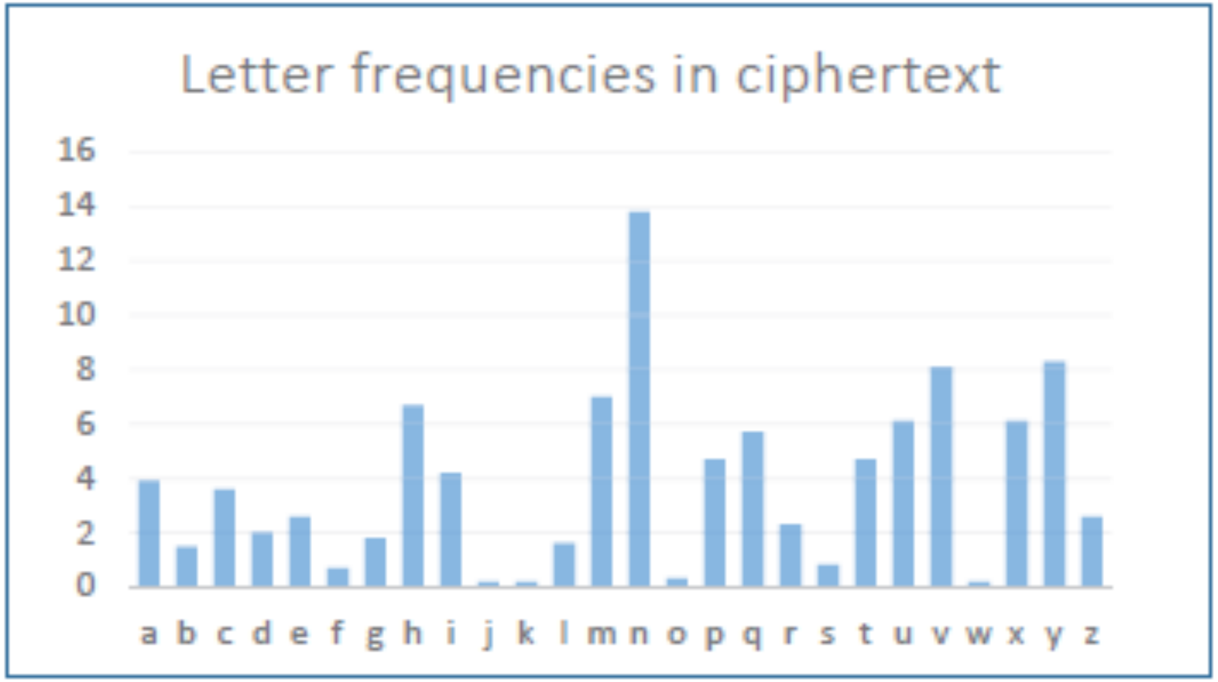
Substitution Cipher

- Superset of shift ciphers: each letter is substituted for another one.
- One implementation: Add a secret key
- Example
  - Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
  - Cipher: ZEBRASCDFGHIJKLMNOPQTUVWXY
- “state-of-the-art” for thousands of years

Monoalphabetic Substitution Cipher

- What is the key space?
  - 26! approx. = 2^88
- Launching an attack?
  - frequency analysis: the study of frequency of letters or groups of letters (grams).
  - Common letters: T, A, E, I, O
  - Common 2-letter combinations (bi-grams): TH, HE, IN, ER
  - Common 3-letter combinations (tri-grams): THE, AND, ING.





Cryptanalysis of Monoalphabetic Substitution

- Dominates cryptography through the first millennium

- Frequency analysis
  - Remember Al-Kindi from 800 AD?

- Lessons?
  - Use large blocks: instead of replacing ~6 bits at a time, replace 64 or 128 bits
    - Leads to block ciphers like DES and AES
  - Use different substitutions to prevent frequency analysis
    - Leads to polyalphabetic substitution ciphers and stream ciphers

Vigenère Cipher (1596)

- Main weakness of monoalphabetic substitution ciphers:
  - Each letter in the ciphertext corresponds to only one letter in the plaintext
- Polyalphabetic substitution cipher
  - Given a key $K = (k_1, k_2, \ldots, k_m)$
  - Shift each letter *p* in the plaintext by k_i, where *i* is modulo *m*

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Plaintext	CRYPTOGRAPHY	
Key	LUCKLUCKLUCK	(Shift 11 20 2 10 11 20 2 11 …)
Ciphertext	NLAZEIIBLJJI	

Kasisky Test

Plaintext	T H E S U N A N D **T H E** M A N I N **T H E** M O O N
Key	K I N G K I N G K **I N G** K I N G K **I N G** K I N G
Ciphertext	D P R Y E V N T N **B U K** W I A O X **B U K** W W B T

Distance = 8

- Repeating patterns (of length >2) in ciphertext are a tell
  - Likely due to repeated plaintext encrypted under repeated key characters
  - The distance is likely to be a multiple of the key length

Cryptanalysis of Vigenère Cipher

- Cracking Vigenère (1854 or 1863)
  1. Guess the key length x using Kasisky test of index of coincidence
  2. Divide the ciphertext into x shift cipher encryptions
  3. Use frequency analysis on each shift cipher

- Lessons?
  - As key length increases, letter frequency becomes more random
  - If key never repeated, Vigenère wouldn't be breakable!

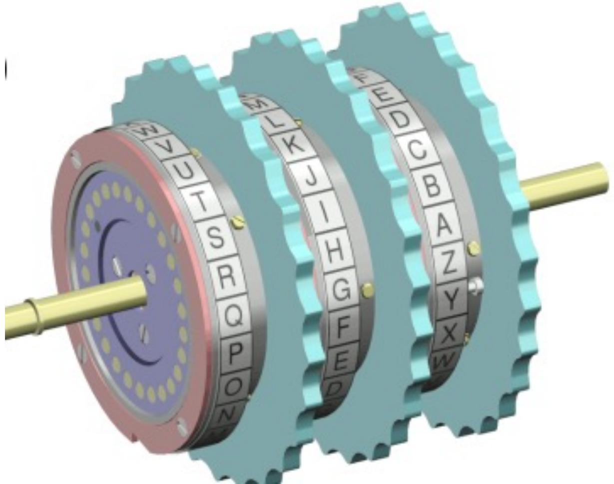
- WW2 German Enigma machine
  - Polyalphabetic substitution cipher
    - Substitution table changes from character to character
    - Rotors control substitutions

- Allies broke Enigma (even before the war), significant intelligence impact

- Computers were built to break WW2 ciphers, by Alan Turing and others

Enigma Machine

- Use rotors that change position after each key
- Key: initial setting of the rotors
- Key space?
  - 26^n for n rotors
- KeyGen:
  - Choose rotors, rotor orders, rotor positions, and plugboard settings
  - 158,962,555,217,826,360,000 possible keys!



Cryptanalysis: Enigma

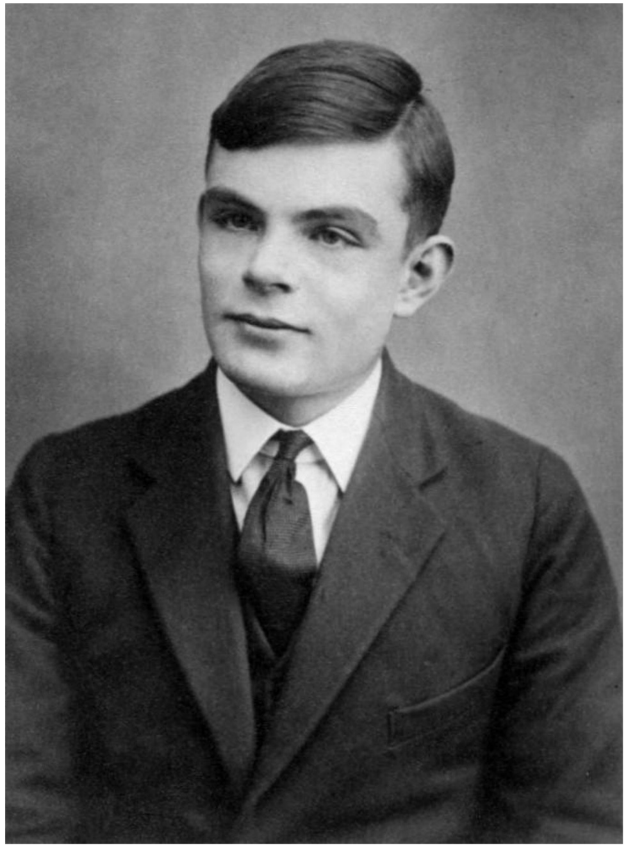
- Polish and British cryptographers built BOMBE, a machine to brute-force Enigma keys
- Why was Enigma breakable?
  - Kerckhoff's principle: The Allies stole Enigma machines, so they knew the algorithm
  - Known plaintext attacks: the Germans often sent predictable messages (e.g. the weather report every morning)
  - Chosen plaintext attacks: the Allies could trick the Germans into sending a message (e.g. "soldiers at Normandy")
  - Brute-force: BOMBE would try many keys until the correct one was found

BOMBE machine

Legacy of Enigma

- Alan Turing, one of the cryptographers who broke Enigma, would go on to become one of the founding fathers of computer science

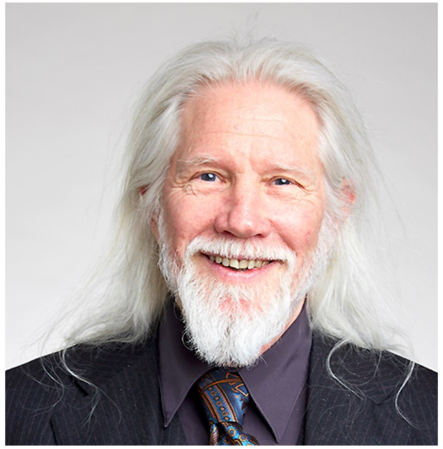
- Most experts agree that the Allies breaking Enigma shortened the war in Europe by about a year

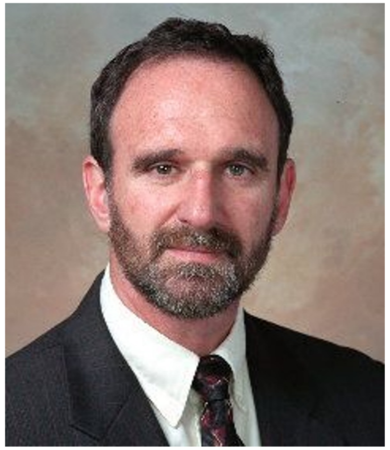


Alan Turing

Cryptography by Computers

- The modern era of cryptography started after WWII, with the work of Claude Shannon
- "New Directions in Cryptography" (1976) showed how number theory can be used in cryptography
  - Its authors, Whitfield Diffie and Martin Hellman, won the Turing Award in 2015 for this paper
- *This is the era of cryptography we'll be focusing on.*





How Cryptosystems work today

- Layered approach:
  - Cryptographic protocols (e.g., CBC mode encryption)
  - Built on: Cryptographic primitives (block ciphers)
- Flavors of cryptography:
  - Symmetric: private key
  - Asymmetric: public key
- Public algorithms: Kerckhoff's principle
- *Security proofs based on assumptions (not this course)*
- Warning!
  - careful about inventing your own!
  - Use vetted libraries to apply crypto algorithms!

Cryptosystem Stack

- Primitives:
  - AES/DES
  - ESA / ElGamal / Elliptic Curve
- Modes
  - Block mode (CBC, ECV, CTR, GCM..)
  - Padding structures
- Protocols:
  - TLS, SSL, SSH
- Usage of protocols:
  - Browser security
  - Secure remote logins

Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.
- Asymmetric cryptography
  - Each party creates a public key p_k and a secret key s_k
  - Inventors won Turing Award!

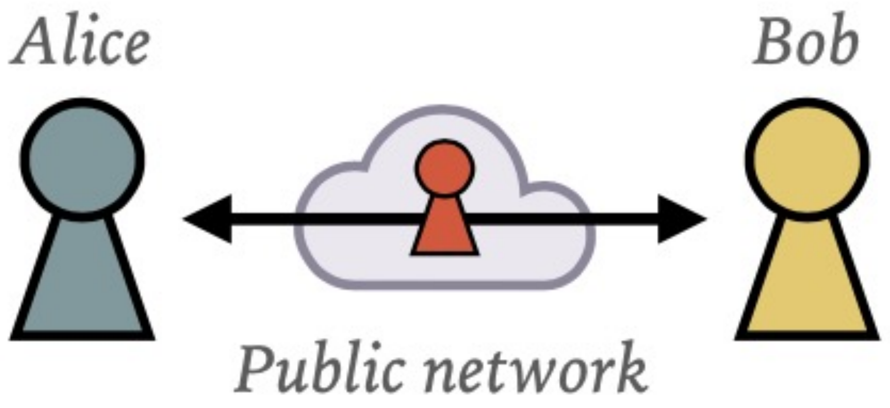
Kerckhoff’s Principle

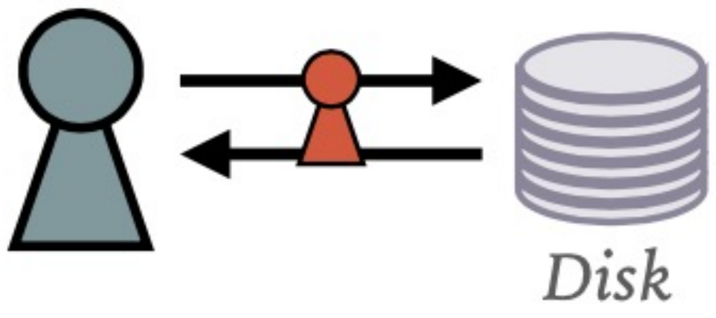
- Security of a cryptographic object should depend only on the secrecy of the secret (private) key.
- Security should not depend on the secrecy of the algorithm itself.
- Foreshadow: Need for randomness – the key to keep private

Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.
  - Challenge: How do you privately share a key?
- Asymmetric cryptography
  - Each party creates a public key p_k and a secret key s_k
  - Challenge: How do you validate a public key?
- Key Insight: Randomness –
  - something an adversary won't know, can't predict and can't figure out.

Randomness





Explicit Uses of Randomness:

- Generate secret cryptographic keys
- Generate random initialization vectors for encryption

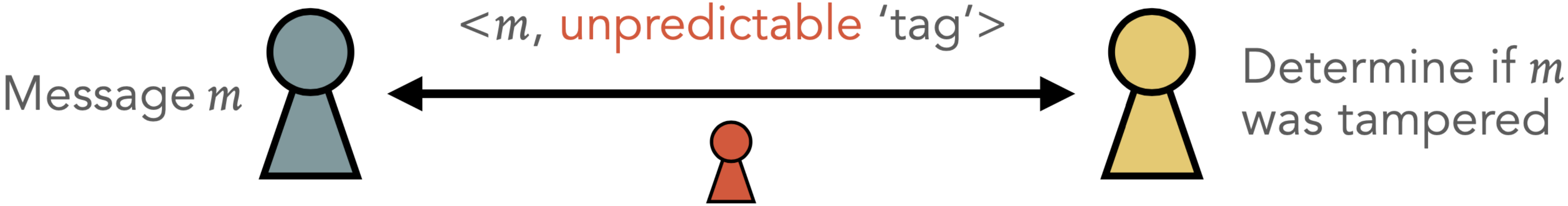
Non-obvious Use cases

- Generate passwords for new users
- Shuffle the order of votes in an electronic voting machine
- Shuffle cards etc. (for online games)

Randomness


Something that leaks
no information about m
Message m
Original m

Randomness



- Ideally, to the attacker, it is indistinguishable from a string of bits chosen uniformly, at random.
- However, this is impossible with Alice and Bob having a shared secret.

What we have, ideally: Random Functions

Consider the set of all permutations $f_i : X \to X$

f_1	0	1	2	3	4	…
f_2	1	0	2	3	4	…
⋮						
$f_{\|X\|!}$	7	9	5	1	8	…

Think of X as all 128-bit bit strings

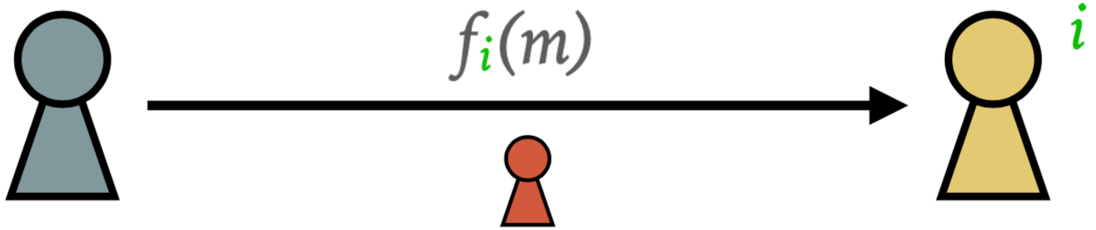
If you know i, then $f_i(x)$ is trivial to invert

If you *don't* know i, then $f_i(x)$ is one-way

"One-way trapdoor function"

Shared secret: index i chosen u.a.r.

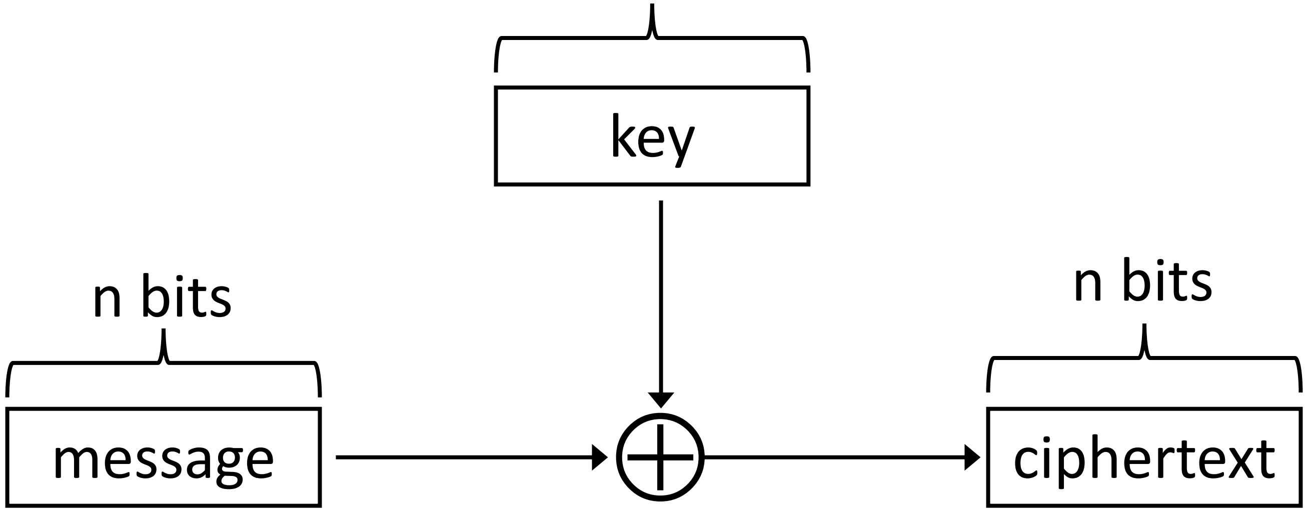
Without knowing i, learns nothing about m



i is our key

One Time Pad (1920s)

- Fix the vulnerability of the Vigenère cipher by using very long keys
- Key is a random string that is at least as long as the plaintext
- Similar encryption as with Vigenère (different shift per letter)



Review: XOR

The XOR operator takes two bits and outputs one bit:

$0 \oplus 0 = 0$
$0 \oplus 1 = 1$
$1 \oplus 0 = 1$
$1 \oplus 1 = 0$

Useful properties of XOR:

$x \oplus 0 = x$
$x \oplus x = 0$
$x \oplus y = y \oplus x$
$(x \oplus y) \oplus z = x \oplus (y \oplus z)$
$(x \oplus y) \oplus x = y$

Review: XOR Algebra

Algebra works on XOR too

$y \oplus 1 = 0$	Goal: Solve for y
$y \oplus 1 \oplus 1 = 0 \oplus 1$	XOR both sides by 1
$y = 1$	Simplify with identities

One-Time Pads: Key Generation

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1

The key K is a randomly-chosen bitstring.

Recall: We are in the symmetric-key setting, so we'll assume Alice and Bob both know this key.

One-Time Pads: Encryption

The plaintext *M* is the bitstring that Alice wants to encrypt.

Idea: Use XOR to scramble up *M* with the bits of *K*.

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1
M	1	0	0	1	1	0	0	1	0	1	0	0

One-Time Pads: Encryption

Alice

K	0	1	1	0	0	1	0	1	0	1	1	1
	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
M	1	0	0	1	1	0	0	1	0	1	0	0
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
C	1	1	1	1	1	1	0	0	0	0	1	1

Encryption algorithm: XOR each bit of *K* with the matching bit in *M*.

The ciphertext *C* is the encrypted bitstring that Alice sends to Bob over the insecure channel.

One-Time Pads: Decryption

Bob receives the ciphertext *C*. Bob knows the key *K*. How does Bob recover *M*?

Bob

K	0	1	1	0	0	1	0	1	0	1	1	1
C	1	1	1	1	1	1	0	0	0	0	1	1

One-Time Pads: Decryption

Bob

K	0	1	1	0	0	1	0	1	0	1	1	1
	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
C	1	1	1	1	1	1	0	0	0	0	1	1
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
M	1	0	0	1	1	0	0	1	0	1	0	0

Decryption algorithm: XOR each bit of K with the matching bit in C.

Cryptanalysis of OTP

- The key is random, so ciphertext is also random
- OTP achieves Perfect Secrecy
  - Shannon or Information Theoretic Security
  - Basic idea: ciphertext reveals no "information" about plaintext

- The adversary believes the probability that the plaintext is m is P(PT=m) **before seeing the ciphertext**
  - Maybe they are very sure, or maybe they have no idea.
- The adversary believes the probability that the plaintext is m is P(PT=m | CT=c) **after seeing that the ciphertext is c.**
  - P(PT=m | CT=c) = P(PT = m) means that after knowing that the ciphertext is *c*, the adversary's belief does not change.
- Intuitively, the adversary learned **nothing** from the ciphertext

Put Another Way

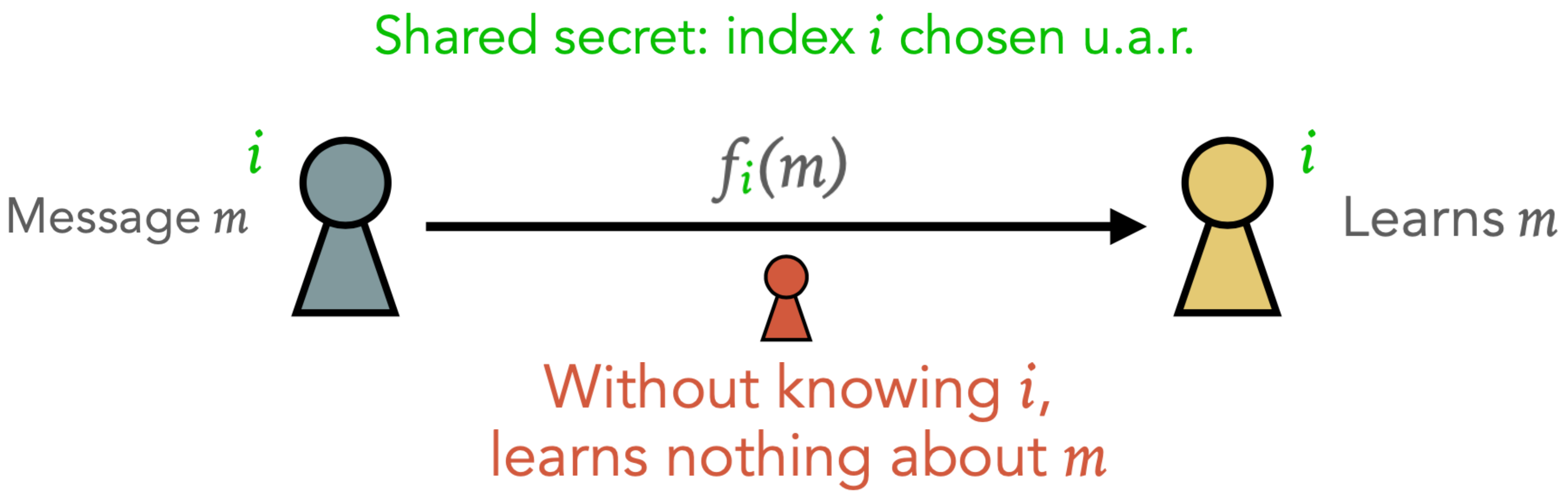
- Imagine you have a ciphertext *c* where the length $|c| = 1000$
- I can give you a key k_i with $|k_i| = 1000$ such that:
  - The decrypted message m_i is the first 1000 characters of Hamlet
- Or, I can give you a key k_j with $|k_j| = 1000$ such that:
  - The decrypted message m_j is the first 1000 characters of the US Constitution

- If an algorithm offers perfect secrecy then:
  - For a given ciphertext of length *n*
  - All possible corresponding plaintexts of length *n* are possible decryptions

Cryptanalysis of OTP

- Intuitively, the key is random, so ciphertext is also random
- OTP achieves Perfect Secrecy
  - Shannon or Information Theoretic Security
  - Basic idea: ciphertext reveals no “information” about plaintext
- Caveats
  - If the length of the OTP key is less than the length of the message…
    - It’s not a OTP anymore, not perfectly secret!
  - If you reuse the OTP key…
    - It’s not a OTP anymore, not perfectly secret!
- Major issue with OTP in practice?
  - How to securely distribute the key books to both parties

What we have, ideally: Random Functions



i is our key

In essence, this protocol is saying "Let's use the ith permutation function"
Infeasible to store all permutation functions – so instead cryptographers construct *pseudorandom functions*

What we have, ideally: Random Functions

- When describing algorithms, we assume access to uniformly distributed bits/bytes to use for key generation
- Where do these actually come from?
- Precise details depend on the system
    - Linux or unix: /dev/random or /dev/urandom
    - Do not use C's rand() or java.util.Random
    - Use crypto libraries instead

Random-number generation

- Two steps:
  1. Continually collect a “pool” of high-entropy (i.e., “unpredictable”) data
  2. When random bits are requested, process this data to generate a sequence of uniform, independent bits/bytes
     - May “block” if insufficient entropy available

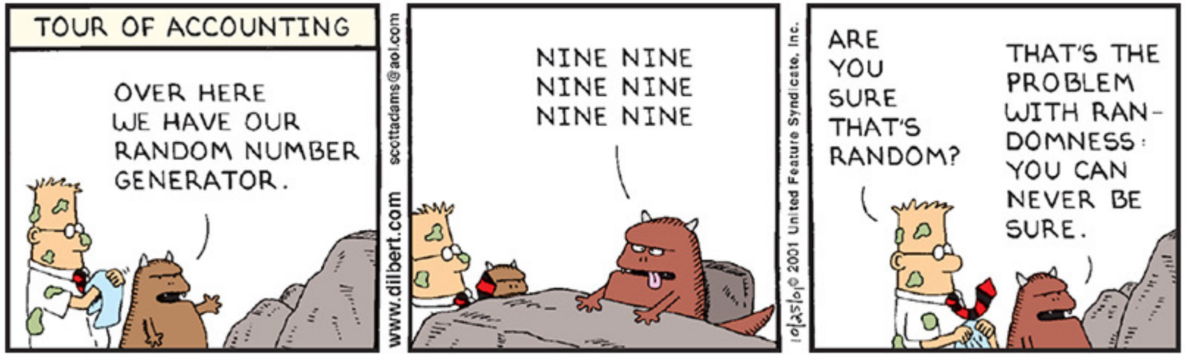
How Random is "Random"?

0XFFFFFFFF EVERY TIME IS 0XDEADBEEF —

How a months-old AMD microcode bug destroyed my weekend [UPDATED]

AMD shipped Ryzen 3000 with a serious microcode bug in its random number generator.

JIM SALTER - 10/29/2019, 7:00 AM



› Uptime | IT and business computing insights

Crypto shocker: four of every 1,000 public keys provide no security (updated)

By Dan Goodin | Published 7 days ago

ORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS | OPINION | A

Flaw Found in an Online Encryption Method

JOHN MARKOFF
blished: February 14, 2012

SAN FRANCISCO — A team of European and American mathematicians and cryptographers have discovered an unexpected weakness in the encryption system widely used worldwide for online shopping, banking, e-mail and other Internet services intended to remain private and secure.

RECOMMEND
TWITTER
LINKEDIN
COMMENTS (127)
E-MAIL

How might we get “good” random numbers?

- For security applications, want “cryptographically secure pseudorandom numbers”
- Libraries include cryptographically secure pseudorandom number generators (CSPRNG)

- Linux:
  - /dev/random: blocking: waits for enough entropy
  - /dev/urandom: nonblocking, possibly less entropy
  - getrandom() – syscall! – by default blocking
- Internally:
  - Entropy pool: gathered from multiple sources
    - e.g.: mouse/keyboard/network timings
- Better idea:
  - AMD/Intel’s on-chip random number generator: RDRAND
  - Hopefully no hardware bugs!